↑P
↑P̧M↑—

*** Kahle ***

A:>cm>defs>f200>chip-specification.lisp.139

1/19/87 19:55:52

```
;;; -*- MODE: TEXT; PACKAGE: CME; MUSER: T; BASE: 10; SYNTAX: ZETALISP; -*-
;;;> ****************************************************************************
;;;> (c) 1985 Thinking Machines Corporation of Cambridge, Massachusetts.
;;;> All rights reserved
;;;>
;;;> This notice is intended as a precaution against inadvertent publication
;;;> and does not constitute an admission or acknowledgement that publication
;;;> has occurred or constitute a waiver of confidentiality.  The Connection Machine Chip
;;;> is the proprietary and confidential property of Thinking Machines Corporation.
;;;> ****************************************************************************
#|
```

Chip Specification for a Connection Machine Chip (version Beta)

{July 1985}
-brewster, danny, george and guy

This document describes the functions of the Connection Machine Chip (version Beta). It describes everything a programmer and board designer will have to know about the functions of the chip. The exact implementation is not described here. To program the router more information is needed than that contained in this document; this will be left to some other document.

The chip contains 16 processors and a router. Each processor is a one-bit slice with an ALU, flag memory, and access paths to the off-chip memory. All processors receive instructions from a common stream supplied externally on pins I-0 through I-16.

The chip provides two different mechanisms for communication among processors: the NEWS connections and the router. The NEWS connections allow each processor to communicate directly with its immediate neighbors in 1, 2, 3 or 4 dimensions. The router is an interface to a packet-switched communication network.

Change log for versions:
Major changes from version Alpha include the elimination of the NEWS pins, the addition of error correction logic, a LOADI and a RUG operation, merging of the Global, Input and Led pins, and improved timing. There are also improvements in the router. The chip is capable of operation in "parity mode" for partial compatibility with version Alpha chips.

```
;;; ****************** CHANGE LOG *****************
;;;
;;; 10/26/85 16:58:49 danny:  Created.
;;;
;;; 11/02/85 11:42:54 danny:  Flip only happens on store and rug-r-x
;;;
;;; 11/02/85 11:43:47 danny:  Define what happens when an error occures while
;;;      referencing an error latch in the rug. The latch ignores the error.
;;;
;;; 11/02/85 13:51:18 danny:  Rug-r-a and Rug-r-c now load ALU-SUM and COND latches.
;;;
;;; 11/06/85 23:58:03 danny:  Documentation on RUG-ACC and note on accumulation.
;;;
;;; 11/07/85 16:05:55 george: Added s-match and s-match-clr Snarf encodings.
;;;
;;; 11/14/85 23:03:44 danny:  flush accummulate, flush run modes, update flipper
;;;
;;; 11/14/85 23:21:41 danny:  replace rug-acc with rug-news
;;;
;;; 11/14/85 23:47:38 danny:  merged in gls's news spec
;;;     but without FSEL and 2 (not 4) FLIP registers
;;;
;;; 11/15/85 02:22:03 danny:  fixed wrong numbers in error table
;;;
;;; 11/15/85 15:13:58 george: fixed numerous small mistakes
;;;     Flushed i/o flag (i/o rug bit remains).
;;;     Updated chip latches table.
;;;     Changed to even parity everywhere.
```

```
;;;    Moved :Signal-Error field to :Error register.
;;;    Added Message-p register.
;;;    Flushed stuff at end into >cm>beta>unimplimented-ideas.text.
;;;
;;; 11/15/85 22:01:16 danny:  updated which-dims-* in tables an in rug
;;;
;;; 11/16/85 16:27:00 danny:  i-parity-error is not checks on reads
;;;
;;; 11/18/85 11:36:49 George:  expanded error code and flush signal-error
;;;     corrected spelling error: which-dims-recieve --> which-dims-receive
;;;
;;; 11/20/85 00:04:44 danny:  wrote more on signal error.
;;;
;;; 12/04/85 11:56:44 George:  made led-cs drve consistant and fix c latch spec
;;;
;;; 1/06/86 12:35:36 danny: The Flipper does permute the data
;;; written to memory on rug-r-a and rug-r-c cycles.
;;;
;;; 1/06/86 12:35:36 danny: added ecc5 input
;;;
;;; 1/07/86 10:25:54 danny:  documented PMODE bit of instruction
;;;
;;; 1/07/86 11:03:39 danny:  updateing timing diagram
;;;
;;; 1/14/86 09:23:09 george: added LOADI RUG-R-A sequence
;;;
;;; 3/05/86 12:34:43 danny:  documented format of rug dimension-n
;;;
;;; *************** END OF CHANGE LOG ***************
```

PROCESSOR DATA PATHS

This section describes the major latches and data paths. The state bits of the machines are divided among "latches," "registers," and "Flags." The flags are temporary state bits associated with each processor, and are explicitly written during the Store Cycle of an instruction sequence. The latches are used for special purposes and are implicity written during the execution of various instructions (see table xxx). The registers hold status information associated with the entire chip (see Table xxx). The a bank of status registers is called the Rug (see RUG section), and it is written explictly on a RUG-W-A or RUG-W-C Cycle.

Each processor is intended to be associated with an external memory.

In a typical bit operation (see the Sequences Section) each processor's ALU takes five single bit inputs. Two of these inputs (A and B) come from the external memory. The third comes from one of the processor's internal flags (called F). There are two special inputs to the ALU called the COND input, which is loaded from a flag, and the C input which is usually loaded with the same thing as A. Each processor has two outputs: Sum (written to the external memory) and Carry (written to a flag). The Cond flag determines whether the sum and carry are written with the newly computed values or rewritten with their previous values.

ALU

The ALU on the chip is composed of 16 single-bit ALUs, one for each processor on the chip. Each one-bit ALU does the same operation on the bits supplied by the input latches. Each processor has different data.

A normal operation takes place in several cycles. During the load cycles of an instruction cycle (LoadA and LoadB) each processor's ALU input latches are read from a flag into F, another flag into COND (can be inverted by an instruction), and one bit from each of any two memory locations into A and B. The ALU produces its two results, Carry and Sum, on the store cycle. The store cycle specifies what flag to update with the carry and what memory location to update with the Sum. Usually the memory location for writing is the same as was read to load the A latch. During the store cycle, if a processor's COND latch contains 1, the Sum output is written back into memory and the Carry goes to a flag. If the COND latch contains 0, then the ALU writes the contents of the C latch back to memory, and does not update the write flag. Therefore, nothing changes.

The SUM output of the processors passes through a permutation device called the flipper, that determines which output goes to which bit of memory. This is described in flipper section.

The ALU can compute any boolean function of its three inputs for each of its two outputs. These two functions are specified by two 8-bit truth tables. The truth table for the carry is specified by the ALU-CARRY field of the LOADA instruction. The truth table for the sum is specified by the ALU-SUM field of the LOADB instruction. The truth tables are constructed so they will produce the correct value when indexed as follows: The a-input becomes the high order bit of a three bit index, the b-input becomes the middle bit and the flag-input becomes the low order bit in the index. The selection of these bits is determined as shown in Table 1. For example, the truth table for the function that computes the logical AND of A and B would be 11000000 (binary).

Table I.

A-Input	B-Input	Flag-input	Carry-Output	Sum-Output
0	0	0	ALU-CARRY-0	ALU-SUM-0
0	0	1	ALU-CARRY-1	ALU-SUM-1
0	1	0	ALU-CARRY-2	ALU-SUM-2
0	1	1	ALU-CARRY-3	ALU-SUM-3
1	0	0	ALU-CARRY-4	ALU-SUM-4
1	0	1	ALU-CARRY-5	ALU-SUM-5
1	1	0	ALU-CARRY-6	ALU-SUM-6
1	1	1	ALU-CARRY-7	ALU-SUM-7

There are also several latches in the machine that are used for special purposes. Data from the router is stored in the RBO latch before it goes to memory for buffering. Data from memory is stored first in the RBP latch and then in the RBI latch as it goes to the router.

Processor Flags

Each processor has associated with it 4 general read/write flags and 4 additional special purpose flags. The general flags are used to store single-bit temporary values, such as the carry during a serial addition operation.

Addr	Name	Write?	Comments
0	Flag0	yes	
1	Flag1	yes	
2	Flag2	yes	
3	Flag3	yes	
4		no	Reserved
5		no	Reserved
6	Com-E	Yes	Writing this will strobe what was read in on the last LoadB cycle into the request latch in the router. Data read from it is the grant output of the router.
7	zero	Yes	Always zero. Writing to Zero has no effect.

The effect of writing non-writable flags is undefined, and reserved for future expansion.

Chip Latches:

This section is a quick reference section. These latches and busses are discussed in the rest of this document.

Control of the ALU Latches is as follows:

```
Name            Bus in    Number  Loading Clock              Comments
A               I         16      (loadI)                    ;;The A Latch is the
                                                             ;; A input to the ALU
                M         16      ((or loadA
                                       rug-w-c))             ;;On rug-w-c,
                rug       16      (rug-r-a)                  ;; the A latch is written by memory.
                                                             ;;On rug-r-a the A latch is written by the rug.
B               M         16      (LoadB)                    ;;B input to ALU
                Cube-In 16        ((and loadA BSel=1 Which-Dims-Send19)
                Route bus 16      ((and loadA BSel=1 Which-Dims-Send=19)
C               M         16      ((or    loadA              ;;For rewriting to memory if
                                          loadI              ;; it is a conditional operation
                                          rug-w-a))          ;; AND the condition failed.
                Rug               (rug-r-c)                  ;;It is also used for accessing
                                                             ;; the rug.
F               FLAG      16      (loadA)                    ;;Flag input to ALU. One
                                                             ;; for each processor.
Cond            (xor                                         ;;Input to ALU:
                 Flag                                        ;; Conditionalization bits
                 inv)     16      (loadB)                    ;;The INV signal is in
                                                             ;; the instruction.
                CS        16      (loadI)                    ;;This is for direct writing
                1s        16      (rug-r-c)
                0s        16      (rug-r-a)

R               M         16      ((or    LOADB              ;;Data from processors to router
                                          Rug-News))

Alu-sum         I         8       (loadb)                    ;;ALU control for sum output.
                Constant8         ((or rug-r loadi))         ;;This is for writing memory
Alu-Carry       I         8       (LoadA)                    ;;ALU control for carry output
Cube-In         M         16      (or Rug-News rug-cube)
                cube pins 16?     (And LatchR
                                       (not rug-news)
                                       (not route))
I/O             I/O pin 1         (LoadA)
Edge-latch      I<7>      1       (Store)                    ;;used in the news circuit
Cube-sel        See store op
                          4       (store)                    ;;used in the news circuit
CS              led-CS-pin                                   ;;This is the Chip Select line.
                          1       (or loadi
                                      read
                                      rug)
```

Other Latches not in the Rug.
(the ones in the rug have been described in the rug section)

```
;EMPTYP    1            Router Empty bit
;RBO       8            Router data going to memory for buffering
;RBI       8            Router data going to router from buffering
;RBP       8            Router data coming from memory from buffering
```

The Basic Cycles

Each time a pulse is applied to the Clock pin, a cycle is executed (see clocking section). The OP Pins and the Instruction Pins determine which type of cycle is executed. This section describes the different OP cycles and their effect on the state of the chip and the driving of the pins.

There are eight possible cycles, as shown in the table below. The RUG cycle is further decoded into six different operations RUG-W-A, RUG-W-C, RUG-R-A, RUG-R-C, RUG-NEWS, RUG-CUBE.

OP NAME	0 NOP	1 LOADA	2 LOADB	3 STORE	4 READ	5 LOADI	6 ROUTE	7 RUG
I0	DC	FLAGR0	COND0	FLAGW0	Z	DATA0	CYCLE0	R/W
I1	DC	FLAGR1	COND1	FLAGW1	Z	DATA1	CYCLE1	A/C
I2	DC	FLAGR2	COND2	FLAGW2	Z	DATA2	CYCLE2	NEWS
I3	DC	BSEL	INV	MB0	Z	DATA3	CYCLE3	MB0
I4	DC	ALUC0	ALUS0	MB0	Z	DATA4	CYCLE4	REG0
I5	DC	ALUC1	ALUS1	MB0	Z	DATA5	CYCLE5	REG1
I6	DC	ALUC2	ALUS2	MB0	Z	DATA6	CHECK0	REG2
I7	DC	ALUC3	ALUS3	EDGE	Z	DATA7	CHECK1	REG3
I8	DC	ALUC4	ALUS4	CUBE0	Z	DATA8	XOR1	REG4
I9	DC	ALUC5	ALUS5	CUBE1	Z	DATA9	XOR2	MB0
I10	DC	ALUC6	ALUS6	CUBE2	Z	DATA10	XOR3	MB0
I11	DC	ALUC7	ALUS7	CUBE3	Z	DATA11	XOR4	MB0
I12	DC	MB0	MB0	MB0	Z	DATA12	SNARF0	MB0
I13	DC	MB0	MB0	MB0	Z	DATA13	SNARF1	MB0
I14	DC	MB0	MB0	MB0	Z	DATA14	SNARF2	MB0
I15	DC	PMODE	PMODE	PMODE	Z	DATA15	ODD	PMODE
I16	DC	PARITY	PARITY	PARITY	Z	PARITY	PARITY	PARITY

DC means Don't-care. MB0 means must be 0. Z means high impedence.

On each cycle, except NOP, the I16 is used for an even parity bit (if the data is all zero, the parity bit will be zero) computed across I0 through I15. We use even parity in order to catch the all ones error, which is more likely in this system. A typical instruction sequence consists of a sequence of several cycles. These are documented in the Sequences section.

Sendr and Latchr are two pins to the chip that are independent of the other cycles, except for Route, Rug-Cube and Rug-News cycles. At any other time Latchr can be asserted and will change some state of the chip. If Latchr is asserted during a Route, Rug-Cube, or Rug-News cycle, the results are not defined.

CYCLE DEFINITIONS:

This section describes what state changes for each of the different op codes. This also lists what pins are used and unused.

NOP CYCLE (OP = 0)

The chip does not change state, (unless latchr is asserted). No errors conditions are checked.

The I/O, Memory, Error, and Instruction pins are undriven.

LOADA CYCLE (OP = 1)

The ALU-CARRY Latch is loaded from the ALUC field. The F Latch is loaded from the flag specified by the FLAGR field. The A Latch and C latch are loaded from the Memory Pins. The I/O latch is loaded from the I/O pin. If BSEL is active, then the B latch is loaded from one of two busses. If Which-Dims-Send = 19 then the B latch is loaded from the route bus otherwise it is loaded from the Cube-In latches.

If the PMODE bit of the instruction or the Parity-p bit of the rug is

active, then the cycle is executed in Parity Mode, otherwise the cycle is execute in ecc mode.

The I/O, LED-CS, Memory, and Instruction pins are undriven. Error is driven if there is an error.

LOADB CYCLE (OP = 2)

The ALU-SUM Latch is loaded from the ALUS field. The COND Latch is loaded from the XOR of the one-bit INV field and the flag specified by the COND field. The B Latch is loaded from the Memory Pins. The R Latch is loaded from the Memory Pins.

If the PMODE bit of the instruction or the Parity-p bit of the rug is active, then the cycle is executed in Parity Mode, otherwise the cycle is execute in ecc mode.

The I/O, Memory, LED-CS and Instruction pins are undriven. Error is driven if there is an error.

STORE CYCLE (OP = 3)

The behavior of the STORE cycle depends on the state of the COND latches. For each processor, if the COND latch holds a one, then the flag specified by the FLAGW field is written by the Carry output of the ALU and the Memory Pin for that processor is driven from the output of the flipper for that processor. If the COND latch for a processor is zero, then no flag is written and the Memory Pin for that processor is driven from the corresponding C latch.

If the PMODE bit of the instruction or the Parity-p bit of the rug is active, then the cycle is executed in Parity Mode, otherwise the cycle is execute in ecc mode.

Other actions of the STORE cycle are independent of the state of the COND latch.

The flipper output is derived by permuting the Sum output from the ALU according to the FLIP0 and FLIP1 registers in the RUG. The NOR of the carry outputs from the ALUs is latched into the Global-latch latch. The EDGE Latch is loaded from the EDGE field. The Cube-Sel Latch is loaded with the XOR of the CUBE field in the instruction and the CUBE-SEL-XOR latches in the RUG. The output of the router is loaded into the RBO latch and the one-bit EMPTYP latch. The RBP latch is loaded into the RBI latch.

The Instruction pins are undriven. The LED-CS Pin is driven from the Global signal. The I/O Pin is driven with the output of the Global-latch latch. The Global Signal (used on the I/O pin) is the output of the Global-latch latch. Error is driven if there is an error.

READ CYCLE (OP = 4)

The chip does not change state on this cycle, unless there is an error while the chip is selected.

If the CS pin is asserted, the Instruction pins are driven with the unpermuted data read from the M-PINS, otherwise they are undriven. The I/O, Memory, and CS-LED pins are undriven. The Error Pin is driven if there is an error and the chip has cs enabled. CS latch is latched.

If the Parity-p bit of the rug is active, then the cycle is executed in Parity Mode, otherwise the cycle is execute in ecc mode.

LOADI CYCLE (OP = 5)

The C latch is loaded from the Memory Pins. The A latch is loaded from the Instruction Pins. The ALU-Sum latch is loaded with a constant that will force the ALU sum output to produce the A input. The COND Latch is loaded with ones if the LED-CS pin is asserted, otherwise with zeros. CS latch is latched.

If the Parity-p bit of the rug is active, then the cycle is executed in Parity Mode, otherwise the cycle is execute in ecc mode.

The I/O, Memory, CS-Led and Instruction pins are undriven. Error is driven if there is an error.

ROUTE CYCLE (OP = 6)

The router is advanced one step, according to the instruction bits, as described in the Router section. The input from the router is read from the R latch, the RBI latch and the Cube latch. The output of the router changes but is not loaded into the RBO latch.

The RBP latch is loaded from Memory pins 8 through 15 (if the ODDP field is one) or Memory pins 0 through 7 (if the ODDP field is zero). Memory pins 0 through 7 (if the ODDP field is one) or Memory pins 8 through 15 (if the ODDP field is zero) are driven from the RBO latch.

This cycle is always executed in Parity Mode.

The I/O and LED-CS pins are driven from the OR of the MSG-P latches. The Instruction pins are undriven. Error is driven if there is an error.

RUG CYCLE (OP = 7)

The Rug instruction is used for reading or writing the rug.

If the PMODE bit of the instruction or the Parity-p bit of the rug is active, then the cycle is executed in Parity Mode, otherwise the cycle is execute in ecc mode. CS latch is latched.

One of five things happens on this cycle, depending on the R/W and A/C bits:

RUG-W-A (R/W=0 A/C=1 NEWS=0)

The rug register specified by the Reg field is written by the A latch. The memory-pins are written to the C latch.

RUG-W-C (R/W=0 A/C=0 NEWS=0)

The rug register specified by the Reg field is written by the C latch. The memory-pins are written to the A latch.

RUG-R-A (R/W=1 A/C=1 NEWS=0)

The rug register specified by the Reg field is read into the A latch. The C latch, permuted by the flipper, drive the memory-pins. ALU-SUM Latch is loaded with :A, so that the Sum output of the alu will read the A Latch (this is needed for rug-r-c, but is also here anyway). The COND Latch is loaded with a zero.
;;it looks like we dont need to disturb the alu-sum or cond latch here because
;;we hop over the ALU anyway in the implementation. -brewster and Dave
;;Douglas 8/25/86

RUG-R-C (R/W=1 A/C=0 NEWS=0)

The rug register specified by the Reg field is read into the C latch. The memory-pins, permuted by the flipper, are written by the A latch. The ALU-SUM Latch is loaded with :A, so that the Sum output of the alu will read the A Latch. The COND Latch is loaded with a one.
;;it looks like we dont need to disturb the alu-sum or cond latch here because
;;we hop over the ALU anyway in the implementation. -brewster and Dave
;;Douglas 8/25/86

RUG-NEWS (R/W=1 A/C=0 NEWS=1)

The R latch and the Cube-In latches are loaded from Memory. Latches A, B,

C are preserved.

RUG-CUBE (R/W=1 A/C=1 NEWS=1)

The Cube-In latches are loaded from Memory. Latches A, B, C and R are preserved.

Other combinations of R/W A/C and NEWS are reserved.

Instruction, global are undriven. Based on r/w the memory is driven or undriven. Error is driven if there is an error.

SEQUENCES SECTION

An operation on a bit in memory takes more than one cycle. The op codes are grouped in sequences for different purposes. Due to the fine granularity of the instruction set for this chip, a great deal of flexibility is gained. There are many other potential sequences than the ones listed here. These are meant to be the common ones and will suffice for most code. All chip features are doable with these sequences. If a programmer needs to try to bum cycles then other sequences might be used. The common sequences are:

LOADB, LOADA, STORE	A two address arithmetic operation
LOADI, STORE	A direct memory write to selected chip
LOADB, LOADA, RUG-W-A, STORE	A three address arithmetic operation
LOADB, LOADA, STORE, ROUTE	A communications step
READ	A direct memory read
RUG-W-A, RUG-W-C	A move from memory to rug register
RUG-R-A, RUG-R-C	A move from rug register to memory
LOADI, RUG-R-C	A direct memory write to all chips
LOADI, RUG-R-A	Copy memory and/or flip memory
LOADI, RUG-W-A	An immediate write to a rug register
LOADA, LOADI, LOADB, STORE	An immediate arithmetic operation
STORE, STORE, STORE, STORE	A 2-D NEWS transmission sequence
STORE, STORE, RUG-R-A, RUG-R-C	Direct Cube Writing
LOADB, (LOADA STORE)*	Single instruction arithmetic
LOADI, (RUG-R-C)*	Clearing memory
LOADB, (LOADA, STORE)*	Unconditional two-address 1 operand (lognot or move)
LOADA, (LOADB, STORE)*	Uncond two-address 1 operand with changing alu-sum instruction (logxor-immediate).

LOADB LOADA STORE:
This is the normal sequence for conditional two-address arithmetic or unconditional three-address arithmetic. If the operation is conditional, then the memory address in the LoadA cycle must be the same as the store cycle. If the operation is unconditional (condition-flag: zero, cond-invert: 1) then this sequence can be used as a three-address instruction where the store cycle has the destination memory address.

LOADI STORE
This is the normal sequence for direct memory write to a chip selected by CS. In other words the data to be written in the selected chip is put on the instruction lines. All chips load the instruction bus into their A latch and a memory location into the C latch on the LoadI. Also, the condition is setup on the selected chip (latched from the CS pin on the LoadI) so that the A latch is brought to the SUM output of the ALU, and on all other chips the C latch is brought to the SUM output. Thus on the store, either the C and A latch to memory.

LOADB LOADA RUG-W-A STORE
This is the normal sequence for conditional three-address arithmetic. The Rug-W-A and the Store must have the same addresses, and the Rug-W-A will load the C latch from memory. The Rug Register will, typically, be Sink.

LOADB LOADA STORE ROUTE
This is the normal sequence for routing. The memory address in the LoadB is sent to the router. The LoadA has its BSEL=ROUTER. The router data comes in the B latch to the ALU.

READ
This is the normal sequence for direct memory read.

RUG-W-A RUG-W-C
This is the normal sequence for writing to a Rug register from memory. The RUG-W-A loads the C latch from memory so the RUG-W-C can write the data into specified register. Writes to multiple rug registers can be pipelined by alternately latching A and C while writing the rug from C and A. This can be used to write N rug registers in N+1 cycles.

RUG-R-A RUG-R-C

This is the normal sequence for reading from a rug register to memory. The RUG-R-A loads the C latch from the rug so the RUG-R-C can write the data into memory. Reads from multiple rug registers can be pipelined by alternately latching A and C while writing the memory from C and A. This can be used to read N rug registers in N+1 cycles.

LOADI RUG-R-C
This is the normal sequence for direct memory write to all chips.

LOADI RUG-R-A
This sequence can be used to copy memory from one address to another on all chips. It can be used to flip memory, since the LOADI sets the C latch with the unflipped source and the RUG-R-A writes the C latch through the flipper to the destination.

LOADI RUG-W-A
This is the normal sequence for immediate write to a Rug register in all chips. The LoadI loads the A latch so the Rug can accept the A for the Rug Write operation. The C latch is loaded from the memory location in the Rug cycle. Additional rug registers may be loaded with the same constant with additional RUG-W-A cycles.

LOADA LOADI LOADB STORE
This is the normal sequence for conditional two-address or unconditional three-address immediate arithmetic, with potentially different immediate values for each processor on a chip. The BSel=Router cannot be used in this case. The address specified by the loadA does not matter. The LoadI address should have the same address of the store if the operation is two-address conditional.

Why this instruction works:

	LoadA	LoadI	LoadB	Store
A	memory	Instr		instr(LoadI)
B			memory	memory(loadB)
C	memory	memory		memory(loadI)
F	flag			flag(loada)
Cond		CS	flag	flag(loadb)
Alu-Sum		mashed	Instr	Instr(loadb)
Alu-Carry	instr			Instr(loada)

What is the sequence for unconditional two-address immediate arithmetic? -brewster and KB0

STORE STORE STORE STORE
This is a normal 2-D NEWS transmission sequence. The 3-D transmission would have 6 store cycles, a 4-D would have 8, and 1-D would only have 1. During each sequence the CubeI field is incremented and the CUBEO field specified, in sequence, each processor on the external boundary of the chip as shown in Table 2. NEWS transmission sequences can be overlapped with a load-load-store sequence to save one cycle. Sendr and Latchr must be strobed on or after each store cycle. ****more on sendr latchr. This section is no good. Please see the news document for how to do news. I hope that is better. -brewster****

STORE STORE RUG-R-A RUG-R-C

[**** looks wrong to me**** -danny]
This is a normal sequence for direct cube writing. This sequence can be overlapped with a load-load-store sequence to save one cycle. Sendr and Latchr must be strobed on or after each store cycle. The data is read from the cube-in latch on the rug-r-a cycle and written to memory on the rug-r-c cycle. ***more on sendr latchr

LOADB (LOADA STORE)*
This is a normal sequence for single instruction arithmetic. The LoadB loads the context and ALU Sum function. This only need happen once for the whole field since a second input to the ALU is not needed for single instruction arithmetic. Such single instruction arithmetic is complement, or complement based on another bit (loaded during the LoadB).

LOADI (RUG-R-C)*
This is a normal sequence for unconditional clearing of memory. The

LoadI is used to set up the function and data and the Rug-R-C's write the same data everywhere. This can be used for setting memory to an arbitrary constant.

LOADB (LOADA STORE)*
This is the normal sequence for unconditional two-address instruction that have only one operand. The only instructions that have this property are lognot (invert a field) and move. Move is accomplished faster by LoadI Rug-W-A.

LOADA, (LOADB, STORE)*
Uncond two-address 1 operand with changing alu-sum instruction. The only instruction that needs this is logxor-immediate. This is done by the instruction ALU field is in the loadB instruction. This can be instructed to toggle the B input to the ALU or not based on a bit in the microcontroller. This is incredibly obscure and only included for hack value.

Error Signaling

Whenever any error occurs, the syndrome register is written with the memory error syndrome (the XOR of the six stored ECC bits and the six computed ECC bits) and the error-code register is written with a number indicating the type of error as shown in table IX. The error-code and syndrome registers remain set until written again, either explictly or as a consequence of another error.

The error pin is driven whenever an error is signalled and the Enable-error register is enabled. All errors are signalable if the SIGNAL-ALL-ERRORS register is high. All errors except single bit memory errors in ecc mode (error code 4) are signalable if the SIGNAL-ALL-ERRORS register is low.

Table IX

```
|#
(defconst cm:*chip-rug-error-codes* '(
;NUMBER  NAME                            COMMENT
(0       :MSG-PARITY-ERROR               "Message Parity Error")
(1       :I-PARITY-ERROR                 "Instruction Parity Error")
(2       :MEMORY-PARITY-ERROR            "Memory Parity Error (in parity mode)")
(3       :CORRECTED-MEMORY-ERROR         "Single Bit Memory Error (in ecc mode)")
(4       :UNCORRECTABLE-MEMORY-ERROR     "Multiple Bit Memory Error (in ecc mode)")
(5       :ILLEGAL-RUG-ERROR              "Write or Read of Illegal Rug-register Error")
))
#|
```

Note: If an error occurs while reading or writing a rug register that is automatically changed by the error system, the automatic change will not take place in that register. This is prevents a register from changing while it is being referenced, which could lead to unpredictable results.

RUG SECTION

The RUG stands for Chip Status Register (RUG). It holds many of the bits in the chip that are not directly associated with processors. This means chip oriented state such as error status, router modes, flip registers, etc. These bits are clumped into registers for fast access. They can be read and written by using the RUG instruction (see instruction section).

There is a pipeline that allows a fast read and write rate from memory. On a rug instruction the a-c bit allows the rug to be read or written (based on another bit) to come from the A or the C latch. The latch that is not going to (or from) the rug is written by (or to) memory. This allows one to write the rug with one latch while loading up the other with the next data to stuff into the rug. This speed is only crucial on some router operations.

RUG REGISTERS

These are read/writable unless otherwise noted.
all signals are active high unless otherwise noted.
setup is all active unless otherwise noted.
number of bits per name is 1 by default.

```
|#
(defvar spec)                                 ;to meta point **temp
(defconst cm:*chip-rug-bits*                  ;interpreted by (cme:parse-rug-registers)
'(
;; Router rug Locations to be stored on trace, reloaded on read
;;      All setup is Don't-care
(0      :empty-p                ;;in snarf
        (:empty-p               :size cm:*chip-number-of-buffers-limit*
                                :setup :enabled
                                :type :spy)
        (:eject-p               :size cm:*chip-number-of-combined-buffers-limit*
                                :setup :enabled
                                :type :status))
(1      :request
        (:request               :type :spy ;;loaded from rug or router computation on some
                                ;;route ops.  (see injector for further details)
                                :setup :DONT-CARE
                                :size cm:*chip-number-of-processors-limit*))
(2      :Match-box
        (:Match-box             :type :spy ;;loaded from rug or router computation on some
                                ;;route ops
                                :size cm:*chip-number-of-match-boxes*)) ;15
,@(cme:heart-rug-state 3)
;; this is a list of rug locations for each dimension (therefore 12 locations).
;; Each rug register has 14 bits (2 for each buffer assuming 7 buffers)
;; the LSB is the vavle-l bit for the top buffer. The merge-l bit is next
;; The setup for each is Don't care because the router will figure it out the
;; first time it is used.  These can be loaded from the rug, or they can be
;; determined by the router on route ops. Assuming 7 buffers and 12 dimensions,
;;(3  :DIMENSION-0 (:DIMENSION-0 :SIZE 14 :SETUP :enabled :TYPE :SPY))
;;(4  :DIMENSION-1 (:DIMENSION-1 :SIZE 14 :SETUP :enabled :TYPE :SPY))
;;(5  :DIMENSION-2 (:DIMENSION-2 :SIZE 14 :SETUP :enabled :TYPE :SPY))
;;(6  :DIMENSION-3 (:DIMENSION-3 :SIZE 14 :SETUP :enabled :TYPE :SPY))
;;(7  :DIMENSION-4 (:DIMENSION-4 :SIZE 14 :SETUP :enabled :TYPE :SPY))
;;(8  :DIMENSION-5 (:DIMENSION-5 :SIZE 14 :SETUP :enabled :TYPE :SPY))
;;(9  :DIMENSION-6 (:DIMENSION-6 :SIZE 14 :SETUP :enabled :TYPE :SPY))
;;(10 :DIMENSION-7 (:DIMENSION-7 :SIZE 14 :SETUP :enabled :TYPE :SPY))
;;(11 :DIMENSION-8 (:DIMENSION-8 :SIZE 14 :SETUP :enabled :TYPE :SPY))
;;(12 :DIMENSION-9 (:DIMENSION-9 :SIZE 14 :SETUP :enabled :TYPE :SPY))
;;(13 :DIMENSION-10 (:DIMENSION-10 :SIZE 14 :SETUP :enabled :TYPE :SPY))
;;(14 :DIMENSION-11 (:DIMENSION-11 :SIZE 14 :SETUP :enabled :TYPE :SPY))
,@(cme:cell-address-rug-state 15)
;;These can be loaded from the rug or the router
;;:Cell-address          24 Bits, bits <3:0> are from
;;                                buffer 0, etc.
;;(15 :CELL-ADDRESS-0 (:CELL-ADDRESS-0 :SIZE 16 :SETUP :DONT-CARE :TYPE :SPY))
;;(16 :CELL-ADDRESS-1 (:CELL-ADDRESS-1 :SIZE 08 :SETUP :DONT-CARE :TYPE :SPY))
```

```
(17     :flip0                    ;;least significant bits, stages 1 and 2
        (:flip0                   :size 16.
                                  :type :status))

(18     :flip1                    ;;most significant bits, stages 4 and 8
        (:flip1                   :size 16.
                                  :type :status))
(19     :error
        (:signal-all-errors       :setup :disabled
                                  :type :status) ;;loaded only from rug
        (:Error-Enable            :setup :enabled ;enable assertion on pin
                                  :type :status)   ;;loaded only from rug
        (:Global-Enable           :setup :enabled ;enable assertion on pin
                                  :type :status) ;;loaded only from rug
        (:Led-Enable              :setup :enabled ;enable assertion on pin
                                  :type :status) ;;loaded only from rug
        (:Syndrome                        ;;the ecc syndrome generated on the most
                                  :size 6              ;;recent memory error
                                  :type :error) ;;loaded on error or by rug op
        (:Error-Code              :size (length cm:*chip-rug-error-codes*) ;6
                                  :type :error) ;;loaded on error or rug op
        )

(20     :message-p
        (:forward-message-p       ;; the message bit for each buffer going forward
                                  :size cm:*chip-number-of-buffers-limit* ;;7
                                  :setup :disabled
                                  :type :spy)
        (:backward-message-p      ;; the message bit for each buffer going backward
                                  :size cm:*chip-number-of-buffers-limit* ;;7
                                  :setup :disabled
                                  :type :spy)
        )

(21     :status                   ;;this must be all read-only except for parity-p
                                  ;;NOTE: parity-p is here so that rug testing is
                                  ;;  more straight-forward.  -George 10/25/85
        (:parity-p                :type :status ;;loaded only from rug
                                  :setup (if cm:*parity-mode*  ;;enables parity mode
                                             :enabled
                                             :disabled))
        (:pin-test-p              :type :status ;;loaded only from rug
                                  :setup :disabled)
        (:input-latch             :type :spy)
        (:global-latch            :type :spy)  ;; Low active to match I/O pin
        ;;Global-latch  NOR of (and condition-bit carry-output) for each
        ;;processor (latched on Store)
        (:cs                      :type :read-only)  ;;chip select pin on the chip
                                                     ;;high active(?)
        (:serial-number           :size 4 :type :read-only)
                                  ;;Betas serial number is 1
                                  ;;(bit 0 a 1, rest 0)
)

(22     :cube-control
                                  ;;Sets up the news direction latches for
                                  ;;off chip and on chip connections
                                  ;;I dont know the number
        (:which-dims-receive      :size 5  ;;which direction of the cube
                                          ;;is to be watched
                                  :type :status ;;loaded only from rug
                                  :setup :enabled)
        (:which-dims-send         :size 5  ;;which direction of the cube
                                   ;;is to be sent out
                                  :type :status ;;loaded only from rug
                                  :setup :enabled)
        (:cube-sel-xor            :size 4  ;;to be xor'd into the CUBE field of STORE
                                  :type :status ;;loaded only from rug
                                  :setup :disabled))
```

```
(23     :router-mode
        (:Fop                     :size 2 ;;bits of operation to the combiner
                                  :setup :disabled ;;this is pass A function
                                  :type :status) ;;loaded only from rug
        (:Backward                :size 1 ;; bit for going forward (0) and backward(1)
                                  :setup :disabled
                                  :type :status)
        (:Inject-P                :size cm:*chip-number-of-injectors-limit*      ;6
                                  :Setup :all-enabled
                                  :type :status)
        (:desperation-route-p     :size cm:*chip-number-of-buffers-limit*
                                  ;;Bit 0 is for the top row of the
                                  ;;heart.  These bits specify whether a row
                                  ;;of the router will desperation route.
                                  :type :status ;;loaded only from rug
                                  :setup ;;all disabled except the last buffer
                                  (append
                                    (loop for buffer below
                                               (1- cm:*chip-number-of-buffers-limit*)
                                          collecting :disabled)
                                    '(:enabled)))) ;;loaded only from rug
(24     :Bad-Wire-Bits
        (:Bad-Wire-Bits           :type :status ;;loaded only from rug
                                  :size cm:*chip-number-of-dimensions*
                                  :setup ;;lower dimensions where there is a machine
                                  ;;dependent on size of the machine.
                                  (append (loop for dimension below
                                               (- cm:*chip-number-of-dimensions*
                                                  (haulong (1-
                                                   cm:*chip-number-of-processors-limit*)))
                                           collecting :disabled)
                                          (loop for dimension from
                                           (- cm:*chip-number-of-dimensions*
                                              (haulong (1-
                                               cm:*chip-number-of-processors-limit*)))
                                           below cm:*chip-number-of-dimensions*
                                           collecting :enabled))))


(25     :parity-errors            ;;parity errors in the router buffers
        (:forward-parity          :size cm:*chip-number-of-buffers-limit* ;;7
                                  :type :spy)
        (:backward-parity                 :size cm:*chip-number-of-buffers-limit* ;;7
                                  :type :spy))
(26     :cube-out
        (:cube-out                :size cm:*chip-number-of-dimensions*
                                  :type :read-only))
(27     :Cube-In
        (:Cube-In                 ;;See chip latches section
                                  :size cm:*chip-number-of-processors-limit* ;16
                                  :type :read-only))

(28     :heart
        (:heart-out               ;;in router.  This need not be setup
                                  ;;NOTE: this should not be put in with another field
                                  ;; which must be preserved while writing this field.
                                  ;; (because LOADA resets heart-out.)
                                  :size cm:*chip-number-of-buffers-limit*
                                  :type :spy)
        (:deliver-p               ;;in router.  This should be disabled on setup
                                  :size cm:*chip-number-of-combined-buffers-limit*
                                  :setup :disabled
                                  :type :spy))
(29     :spread
        (:spread                  ;;inverse of global-latch, repeated
                                  ;;16 times (once for each processor)
                                  :size 16 :type :read-only))
(30     :illegal-30
        (:illegal-30              ;; writing to this will generate and error
```

```
                                ;; reads zeros
                                :size 0
                                :type :illegal-address))
(31     :Sink                   ;;this can be written with no effect
                                ;;NOTE: This must be at address 31 (error-system).
        (:Sink                  ;;Place to dump unwanted bits and read 0's
                                :size 0
                                :type :address-only))
))
#|
```

Spreading

The SPREAD register of the rug allows a value of the inverse of the GLOBAL-LATCH latch (the global signal on the most recent store cycle) to be written to all of the processors. This is useful for copying information among the processors of the chip. The value read by the spread register is not effected by GLOBAL-ENABLE.

The Flipper

The Flipper is a device for permuting the 16 SUM outputs of the 16 processors on the chip, before they are written into memory. It is able to perform various useful permutations (as defined in Table III) including shifting and reversing. The flipper permutes the alu-sum outputs of the alu according to the permutation specified by Flip0 and Flip1 registers of the RUG. This permutes the data written to memory on the store cycles.

Conditionalization interacts with the flipper as follows. Each processor calculates its unpermuted alu sum, without regard to the condition. These values are then permuted by the flipper. Each processor will then write either the permuted value or the contents of the c-latch, depending on the state of the condition.

The action of the flipper is determined by the contents of the Flip0 and Flip1 registers of the RUG. Each bit controls one "exchanger." Whenever the bit is one, the exchange will exachange its two data bits. Whenever the bit is zero, the exchange will pass its two data bits without permuting them. The exchangers are arranged in four stages, with the output of each stage feeding into the input to the next, so that the total action of the flipper is the composition of the actions of the four stages. The correspondence between control bits, exchangers, and stages in shown in Table I. The Table shows the effect of each bit of the register on each to the four stages of the flipper.

Table I

(The entry (a b) in with control bit c and stage s indicates that Bit c of the Flip register controls an exchanger across data bit a and b in stage s.)

```
|#
(defconst cm:*flip-bits* '(
;Control
;Bit    Stage           Exchanger

(0      0               (0 1))
(1      0               (2 3))
(2      0               (4 5))
(3      0               (6 7))
(4      0               (8 9))
(5      0               (10 11))
(6      0               (12 13))
(7      0               (14 15))
(8      1               (0 2))
(9      1               (1 3))
(10     1               (4 6))
(11     1               (5 7))
(12     1               (8 10))
(13     1               (9 11))
(14     1               (12 14))
(15     1               (13 15))
(16     2               (0 4))
(17     2               (1 5))
(18     2               (2 6))
(19     2               (3 7))
(20     2               (8 12))
(21     2               (9 13))
(22     2               (10 14))
(23     2               (11 15))
(24     3               (0 8))
(25     3               (1 9))
(26     3               (2 10))
(27     3               (3 11))
(28     3               (4 12))
(29     3               (5 13))
```

```
(30     3              (6 14))
(31     3              (7 15))
))

#|

Table III

Useful Values for the flipper
|#
(defvar cm:*flip-useful-values* '(
;these are the values used by the diagnostics to test the chip
;name           ;flip0  ;flip1  ;permutation
(:IDENTITY      0       0       (0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15))
(:BACKWARD      43775   32904   (1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 0))
(:ROTATE-14     65280   49356   (2 3 4 5 6 7 8 9 10 11 12 13 14 15 0 1))
(:ROTATE-13     22015   57582   (3 4 5 6 7 8 9 10 11 12 13 14 15 0 1 2))
(:ROTATE-12     0       61695   (4 5 6 7 8 9 10 11 12 13 14 15 0 1 2 3))
(:ROTATE-11     43775   63607   (5 6 7 8 9 10 11 12 13 14 15 0 1 2 3 4))
(:ROTATE-10     65280   64563   (6 7 8 9 10 11 12 13 14 15 0 1 2 3 4 5))
(:ROTATE-9      22015   65041   (7 8 9 10 11 12 13 14 15 0 1 2 3 4 5 6))
(:ROTATE-8      0       65280   (8 9 10 11 12 13 14 15 0 1 2 3 4 5 6 7))
(:ROTATE-7      43775   32648   (9 10 11 12 13 14 15 0 1 2 3 4 5 6 7 8))
(:ROTATE-6      65280   16332   (10 11 12 13 14 15 0 1 2 3 4 5 6 7 8 9))
(:ROTATE-5      22015   8174    (11 12 13 14 15 0 1 2 3 4 5 6 7 8 9 10))
(:ROTATE-4      0       4095    (12 13 14 15 0 1 2 3 4 5 6 7 8 9 10 11))
(:ROTATE-3      43775   1911    (13 14 15 0 1 2 3 4 5 6 7 8 9 10 11 12))
(:ROTATE-2      65280   819     (14 15 0 1 2 3 4 5 6 7 8 9 10 11 12 13))
(:FORWARD       22015   273     (15 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14))
(:WEST          26367   0       (1 2 3 0 7 4 5 6 9 10 11 8 15 12 13 14))
(:EAST          39423   0       (3 0 1 2 5 6 7 4 11 8 9 10 13 14 15 12))
(:NORTH         65535   61695   (7 6 5 4 11 10 9 8 15 14 13 12 3 2 1 0))
(:SOUTH         65535   4095    (15 14 13 12 3 2 1 0 7 6 5 4 11 10 9 8))
(:W             255     0       (1 0 3 2 5 4 7 6 9 8 11 10 13 12 15 14))
(:X             65535   0       (3 2 1 0 7 6 5 4 11 10 9 8 15 14 13 12))
(:Y             65535   255     (7 6 5 4 3 2 1 0 15 14 13 12 11 10 9 8))
(:Z             65535   65535   (15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0))
(:reverse       #xFFFF  #xFFFF  (15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0))
))
#|
```

PARITY and ECC

There are four error detection mechanisms in the chip. Each instruction has parity protection on executed cycles (not unselected read or nop). Memory parity or ECC is always checked when memory is read into the chip. The router can signal an error if message parity is not preserved either going forward or backward. If an error is signaled, and the error pin and/or an error recovery system is activated.

Instruction Parity Errors:
On all cycles execpt READ and NOP, the parity of the instruction is checked. If an error occurs an error is signaled. Even parity is used.

Memory Read Errors:
At any time the chip is either in Parity Mode or in ECC mode, depending on the Parity-p bit in the RUG and whether the route cycle is being executed. When in Parity mode, Even parity is used.

The chip operates in memory parity mode when the parity-p rug bit is asserted or when the chip is executing a route instruction. In parity mode the ECC Pin 0 writes or expects to read an even parity bit for the Memory Pins. ECC Pins 1 through 5 are driven with the same data as Memory Pins 1 through 5, respectively.

The chip operates in ECC mode when the parity-p rug bit is low. The ECC (Error Correcting Code) system on chip corrects all single bit memory errors and detects all double bit memory errors. Each ECC bit is computed by XORing or XNORing eight of the Memory Pins according to the table cm:*ecc-code*.

While changing between parity to ecc, the chip should be referencing a memory value that is valid in both modes, for example 16 (decimal).

Note that memory checking and writing on a ROUTE cycle (the address is the address of a bit slice in the router buffers) is a very special case. For ROUTE, it always reads and writes in Parity mode, regardless of the setting of :Parity-p. Also, memory on ROUTE cycles is written and read in halves; while one half is written, the other half is read. There are seven buffers, so the eighth bit of the half is used to insure that each half is always an even number of bits; hence the parity bit is always zero. For this mechanism to work properly, router buffers must be zeroed in Parity mode before any routing is done.

Router Parity:
Forward router parity works as follows: On the input of the heart parity is generated and can be injected into the message. This is checked on the output of the heart. Parity bits can be placed arbitrarily often in the message and probably in the address as well. Both the address and the data are protected with this scheme. Since the address changes as it progresses from chip to chip, the parity is also updated. This protection gives multiple parity protection of a message and the address. This mechanism can also diagnose what wire went bad by degrading the performance of the petit cycle by 5%. This is accomplished by saving the state of the heart after each petit cycle so that if an error occurred one can examine the decisions made by the heart and find what wire went wrong. This would track all error that occurred over wires during address transmission. If one saved the buffer state also, then the petit cycle could be rerun continuously to try to make the error recur.
Backward router parity is similar, except the parity is placed on the message just after the heart, and checked right before the heart. (this view of the heart is from the forward view).

Signalling an error:

When an error occurs the error-code field in the rug is loaded with the type of error on the following cycle (unless the error register happens to be reference on that cycle, in which case the error-code is not recorded). All errors, except corectable memory-errors when the signal-all-errors latch is low, will also signal an error on the error pin on the cycle following the error, if the enable

```
errro bit is high.

|#
(defconst
 cm:*ECC-CODE*
'(

;                                 Memory Pin
;ECC PIN    0  1  2  3  4  5  6  7  8  9  10 11 12 13 14 15

(XNOR       1  1  1  0  0  0  0  0  0  1  1  0  1  1  0  1)    ;ECC0
(XNOR       0  0  0  1  1  1  0  0  1  0  1  1  0  1  1  0)    ;ECC1
(XNOR       0  0  0  0  0  0  1  1  1  1  0  1  1  0  1  1)    ;ECC2
(XNOR       0  1  1  0  1  1  0  1  1  1  1  0  0  0  0  0)    ;ECC3
(  XOR      1  0  1  1  0  1  1  0  0  0  0  1  1  1  0  0)    ;ECC4
(  XOR      1  1  0  1  1  0  1  1  0  0  0  0  0  0  1  1)    ;ECC5

;(A "1" indicates that the memory pin is used in computing the
;bits 0,1,2 are computed with XNOR, bits 3,4,5 are computed with XOR
;corresponding ECC pin.)
))
#|

Whenever any memory error (correctable or uncorrectable) is detected the
Syndrome register of the RUG is loaded with the error syndrome.
If a single bit error is detected, it is corrected without signalling an
error (unless the Signal-All-Errors rug bit has a 1), and the
Corrected-Memory-Error error code is set in the Error-Code rug field.
If a multiple bit error is detected, the Uncorrectable-Memory-Error
error code is set in the Error-Code rug field.  Any multiple
bit error that generates a syndrome with even parity will be detected.
This includes all two bit errors.
```

SendR, LatchR, Direct Cube Write, and NEWS

The transmission of data between chips is controlled by the SendR and LatchR pins. A chip with its SendR pin active may drive data onto one or more of its cube pins. A chip with its LatchR pin active may latch data from one or more of its cube pins into the corresponding Cube-In latches. (Note that a RUG instruction can also load the Cube-In latches.)

There are three distinct mechanisms from transmitting and receiving data: direct cube write, NEWS, and the router. These three mechanisms are related and share some parts. Only direct cube writing and NEWS are discussed in this section.

Direct cube writing allows processors 0-11 to read and write cube pins 0-11 directly. In this mode the data driven to cube pins is taken from the R latches for processors 0-11, and data from cube pins is read into the Cube-In latches of processors 0-11. (The LOADA instruction with BSEL=Router can then transfer data from the Cube-In latches to the B latches.) Processors 12-15 do not provide data for cube pins, and Cube-In latches 12-15 are always cleared in this mode.

The NEWS mechanism may be used for regular grid-like patterns of communication between processors. The pattern is controlled by a combination of the the News and Flip registers in the RUG and the 4-bit Cube-Sel latch (which is loaded by the STORE instruction). The News section of the RUG contains the Which-Dims-Send register, the Which-Dims-Receive register, and the Cube-Sel-Xor register. The Flip section of the RUG contains 32 bits for controlling the flipper. These registers must be reloaded each time a different NEWS direction or transmission mechanism is to be used.

When the SendR pin is active, the cube pins that are enabled for driving are selected by the Which-Dims-Send field of the Cube-Control register. The action of the Which-Dims-Send field is shown in Table 5. Values 0-11 are used for NEWS communication. Value 19 is used for routing, and value 18 is used for direct cube writes.

```
|#

(defconst cm:*which-dims-send-encoding*
;; Table 5.  Which-Dims-Send
;; Which-Dims-Send | Source of Data | Cube Wires Driven
;;--------------------------------------------------------
'((0                        :R[Cube-Sel]-to-Cube-Pin[0])
  (1                        :R[Cube-Sel]-to-Cube-Pin[1])
  (2                        :R[Cube-Sel]-to-Cube-Pin[2])
  (3                        :R[Cube-Sel]-to-Cube-Pin[3])
  (4                        :R[Cube-Sel]-to-Cube-Pin[4])
  (5                        :R[Cube-Sel]-to-Cube-Pin[5])
  (6                        :R[Cube-Sel]-to-Cube-Pin[6])
  (7                        :R[Cube-Sel]-to-Cube-Pin[7])
  (8                        :R[Cube-Sel]-to-Cube-Pin[8])
  (9                        :R[Cube-Sel]-to-Cube-Pin[9])
  (10                       :R[Cube-Sel]-to-Cube-Pin[10])
  (11                       :R[Cube-Sel]-to-Cube-Pin[11])
  (12                       :reserved)
  (13                       :reserved)
  (14                       :reserved)
  (15                       :reserved)
  (16                       :reserved)
  (17                       :reserved)
  (18                       :R[0-11]-to-Cube-Pin[0-11])
  (19                       :Router-to-Cube-Pin[0-11])
  (20                       :reserved)
  (21                       :reserved)
  (22                       :reserved)
  (23                       :reserved)
  (24                       :reserved)
  (25                       :reserved)
```

```
  (26                   :reserved)
  (27                   :reserved)
  (28                   :reserved)
  (29                   :reserved)
  (30                   :reserved)
  (31                   :none)
))

#|
```

The data coming in the cube pins may be loaded into the Cube-In latch when LATCHR is active. LATCHR is unclocked. In other words, whenever latchr is active a latch is latched. Which cube pins are read, and which bits of the Cube-In latch are loaded, is determined by the Which-Dims-Receive field of the Cube-Control rug register, as shown in Table 7. Values 0-11 are used for NEWS communication. Value 28 is used for reading input from ECC5 pin (for alpha compatibilty). Value 29 is used for reading input from the I/O pin. Value 30 is used for NEWS communication in processors that are at the edge of the grid and should receive a boundary value. Value 18 is used for direct cube writes and for routing.

```
|#

(defconst cm:*which-dims-recieve-encoding*
;; Table 7.  Which-Dims-Receive (5 bits)
;; Which-Dims-Receive | Source Read |  Latches loaded
;;-----------------------------------------------------------
'((0                    :Cube-Pin[0]-to-Cube-In[Cube-Sel])
  (1                    :Cube-Pin[1]-to-Cube-In[Cube-Sel])
  (2                    :Cube-Pin[2]-to-Cube-In[Cube-Sel])
  (3                    :Cube-Pin[3]-to-Cube-In[Cube-Sel])
  (4                    :Cube-Pin[4]-to-Cube-In[Cube-Sel])
  (5                    :Cube-Pin[5]-to-Cube-In[Cube-Sel])
  (6                    :Cube-Pin[6]-to-Cube-In[Cube-Sel])
  (7                    :Cube-Pin[7]-to-Cube-In[Cube-Sel])
  (8                    :Cube-Pin[8]-to-Cube-In[Cube-Sel])
  (9                    :Cube-Pin[9]-to-Cube-In[Cube-Sel])
  (10                   :Cube-Pin[10]-to-Cube-In[Cube-Sel])
  (11                   :Cube-Pin[11]-to-Cube-In[Cube-Sel])
  (12                   :reserved)
  (13                   :reserved)
  (14                   :reserved)
  (15                   :reserved)
  (16                   :reserved)
  (17                   :reserved)
  (18                   :Cube-Pin[0-11]-to-Cube-In[0-15]) ;; Cube-In[12-15] are cleared
  (19                   :reserved)
  (20                   :reserved)
  (21                   :reserved)
  (22                   :reserved)
  (23                   :reserved)
  (24                   :reserved)
  (25                   :reserved)
  (26                   :reserved)
  (27                   :reserved)
  (28                   :ECC5-Latch-to-Cube-In[Cube-Sel])
  (29                   :IO-Latch-to-Cube-In[Cube-Sel])
  (30                   :Edge-Latch-To-Cube-In[Cube-Sel])
  (31                   :none)
))

#|
```

The standard NEWS cycle is then LOADA[BSEL=1] RUG-NEWS {STORE}*, where each pair of STORE cycles has a different value for the CUBE field, and of each pair one asserts Sendr-even/Latchr-odd and the other asserts Sendr-odd/Latchr-even (the second STORE may be a NOP instead, since the important data is latched). The transfer of bits is pipelined by one LLS* cycle. The LOADA copies the Cube-In data just received to the B latch, as well as loading the A latch in the usual manner. The RUG-NEWS reads from memory data to be sent (on-chip and off-chip) and loads it into the R latch and Cube-In latch. Then the

STORE instructions store the data just received (now in the B latch), combined with the A data if desired, into memory (many times, redundantly), and also transfer data across the cube wires, sourced from the R latches and stored into the Cube-In latches for the next cycle. To transfer N bits therefore requires N LoadA[Bsel=1] Rug-News {Store}* cycles followed by a LoadA[Bsel=1] Rug-News {Store} cycle to store the last bit.

Rules for Clocking:

There is only one clock pin on the chip.

When the clock is 0, the other input pins may change arbitrarily without affecting the state of the chip, except Latchr. If latchr is asserted then the latch is opened to secure the data coming in over the cube lines. Before the clock becomes active, all pins must be stable for some setup time.

When the clock is 1, all instruction pins must remain stable. Memory pins may change but must remain stable for time not less than a setup time before the clock becomes Low. Output pins may become driven as early T-CLH-on after the inactive to active transition, but are not guaranteed to reach these correct values until T-CLH-ready after the transition. In the case of output pins that depend directly on input pins in the current cycle, the output will reach their stable state T-CLH-ready after the transition or T-propagate after the corresponding input pins reach their proper states, which is even later. After the clock goes low, all pins must remain stable for at least T-CHL-hold.

The internal control circuitry is designed so that if the chips are wired properly no software initialization error can cause two internal drivers to simultaneously drive the same line.

Initialization:

After power-up, the following steps must be taken in sequence to assure the proper initialization of the processors and router <>

-Initialize the rug (this also initializes the error condition in the rug)
-Write zero to each of processors flags

ELECTRICAL SPECIFICATIONS

Inputs

Outputs

Power Supplies

Pin Summary

Name	Number	I/O	Assertion level	Comment
Control Pins				
Clock	1	Input	High	Strobe to execute instuction
LED-CS	1	In/Out	Low	Chip select, active low and LED
Error	1	Output	Low	Error during current cycle, open drain
I/O	1	In/Out	High	Global output and input, open drain
Processor Instruction Pins				
OP0-2	3	Input	High	What type of instruction
I0-16	16+1	In/Out	High	Instruction and Data bus + parity
Memory Pins				
M0-15	16	In/Out	High	Memory bus Data
ECC0	1	In/Out	High	Error Check Bit 0 or Memory bus Data Parity
ECC1-5	5	In/Out	High	Error Check Bits 1-5
Communications Pins				
Cube0-11	12	In/Out	Low	Cube connection 0-11
LatchR	1	Input	High	Latch router inputs
SendR	1	Input	Low	Send router outputs
Power Pins				
VDD	4	power	High	+5 power
VSS	4	power	Low	ground
Total	68			

PINS

I/O Pin

	I/O	Comments
LoadA	I	Input for i/o latch
LoadB	I	Input for flag read
LoadI	Z	Not specified
Store	O	Output of global signal (low active) unless global enable is inactive
Read	Z	Not specified
Route	O	Output of buffer empty (low active) unless global enable is inactive
NOP	Z	Not specified
RUG	Z	Not specified

The Global signal (the output part of the I/O pin) is used for communication with the host computer and for fast I/O. The logical NOR of each processor's Carry is available on the I/O pin on the Store instruction. On the route instruction the Buffer Empty signal is put on the I/0 pin. That is, if a chip has a router buffer that is NOT empty then it asserts the global.

Led/CS Pin
Very much like the I/O pin.

	Led/CS	Comments
LoadA	x	Not specified
LoadB	x	Not specified
LoadI	CS	Input for setting COND Latch
Store	Led	Output of global signal (low active) unless LED enable is inactive
Read	CS	Input for memory parity checking
Route	Led	Output of buffer empty (low active) unless LED enable is inactive
NOP	X	Not specified
RUG	CS	Input

Led enable	Global	LED
1	0	1 (on the pin is 5V, light is ON)
1	1	0
0	0	0
0	1	0

ERROR PIN

A error is signalled if:
  (and <There is an error>
       Enable-Error)

An error is signalled only once so that additional errors can be detected.

	Enabled	Comments
LoadA	Yes	(or Memory-error Instruction-error)
LoadB	Yes	(or Memory-error Instruction-error)
LoadI	Yes	Instruction-error
Store	Yes	(or Memory-error Instruction-error)
Read	(if cs yes)	Memory-error
Route	yes	(or Memory-error Instruction-error (and enable-message-error message-error))
NOP	no	Never
RUG	yes	(or Memory-error Instruction-error)

The Error pin is always driven directly from the ERROR latch, unless Enable-Error is not asserted.

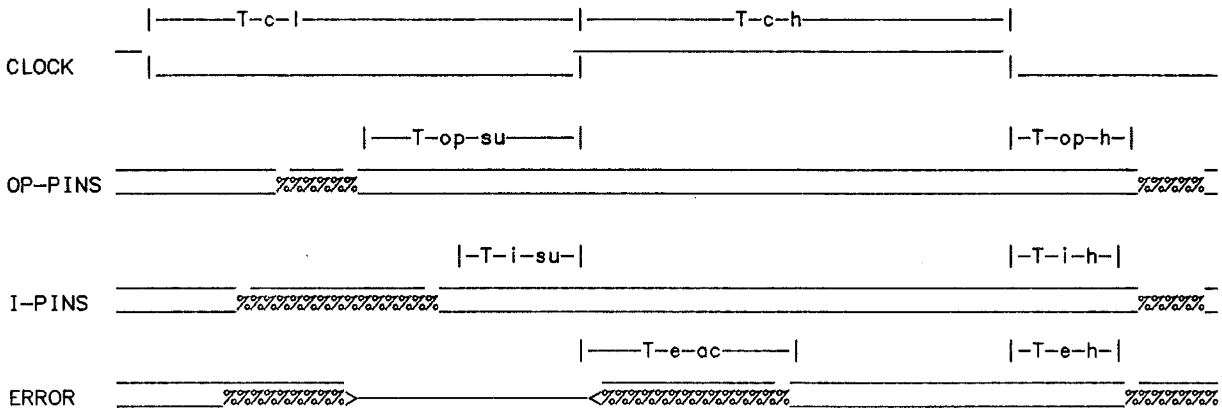
Memory Pins and Instruction Pins

	M I/O	I I/O	Comments

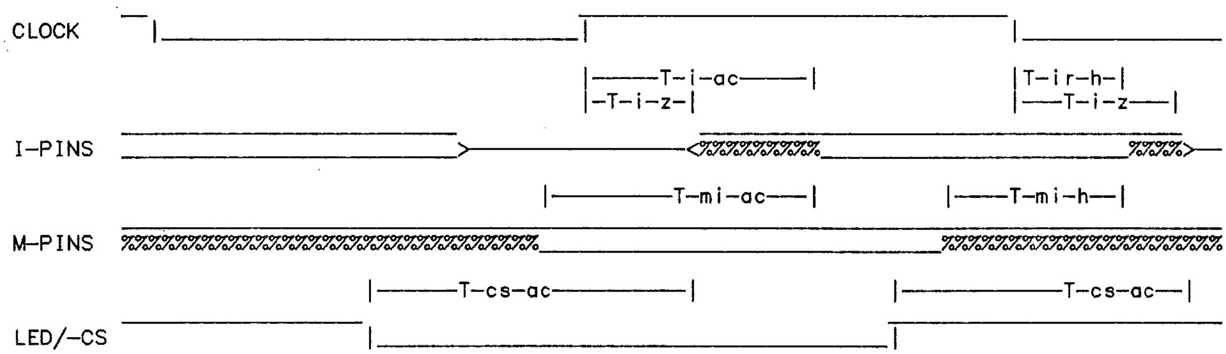
```
LoadA    I        I
LoadB    I        I
LoadI    I        I
Store    O        I
Read     I        O
Route    half I   I          Memory is split, depending on odd bit in
         half O              Instruction.
NOP      X
RUG      I or O   I          Memory is I if Read bit is disabled
                             (write the rug).  Memory is O if Read bit
                             is enabled (read the rug)

Cube Pins
If sendr active (unclocked): output
                        Else input


Pinouts:
|#
(defconst cm:*chip-pinouts* '(

;Name            Pin #       Type           Alpha Version Names

; Top side starting at left looking from component side.
(i-pins[1]        9        :bidirect        i-pins[1])
(i-pins[2]        8        :bidirect        i-pins[2])
(i-pins[3]        7        :bidirect        i-pins[3])
(i-pins[4]        6        :bidirect        i-pins[4])
(i-pins[5]        5        :bidirect        i-pins[5])
(i-pins[6]        4        :bidirect        i-pins[6])
(i-pins[7]        3        :bidirect        i-pins[7])
(i-pins[8]        2        :bidirect        i-pins[8])
(vss              1        :vss             vss)
(vdd              68       :vdd             vdd)
(i-pins[9]        67       :bidirect        i-pins[9])
(i-pins[10]       66       :bidirect        i-pins[10])
(i-pins[11]       65       :bidirect        i-pins[11])
(i-pins[12]       64       :bidirect        i-pins[12])
(i-pins[13]       63       :bidirect        i-pins[13])
(i-pins[14]       62       :bidirect        i-pins[14])
(i-pins[15]       61       :bidirect        i-pins[15])

; Right side starting at top.
(i-pins[16]       60       :bidirect        i-pins[16])
(ECC0             59       :bidirect        M-pins[16])
(M-pins[15]       58       :bidirect        M-pins[15])
(M-pins[14]       57       :bidirect        M-pins[14])
(M-pins[13]       56       :bidirect        M-pins[13])
(M-pins[12]       55       :bidirect        M-pins[12])
(M-pins[11]       54       :bidirect        M-pins[11])
(vdd              53       :vdd             vdd)
(vss              52       :vss             vss)
(M-pins[10]       51       :bidirect        M-pins[10])
(M-pins[9]        50       :bidirect        M-pins[9])
(M-pins[8]        49       :bidirect        M-pins[8])
(cube-pins[11]    48       :bidirect        cube-pins[11])
(cube-pins[10]    47       :bidirect        cube-pins[10])
(cube-pins[9]     46       :bidirect        cube-pins[9])
(cube-pins[8]     45       :bidirect        cube-pins[8])
(cube-pins[7]     44       :bidirect        cube-pins[7])

; Bottom side starting at right.
(cube-pins[6]     43       :bidirect        cube-pins[6])
(ECC2             42       :bidirect        WS-pin)
(ECC1             41       :bidirect        E-Pin)
(ECC3             40       :bidirect        N-Pin)
(ECC4             39       :bidirect        LED-pin)
(Error-pin        38       :output          Error-pin)
(I/O              37       :bidirect        Global-pin)
(vdd              36       :vdd             vdd)
(vss              35       :vss             vss)
```

```
(clock-pin        34        :input          clock-pin)
(ECC5             33        :bidirect       IN-pin)
(Led/CS           32        :bidirect       cslow-pin)
(sendr-pin        31        :input          sendr-pin)
(latchr-pin       30        :input          latchr-pin)
(OP-Pins[2]       29        :input          OP-Pins[2])
(OP-Pins[1]       28        :input          OP-Pins[1])
(OP-Pins[0]       27        :input          OP-Pins[0])

; Left side starting at bottom.

(cube-pins[0]     26        :bidirect       cube-pins[0])
(cube-pins[1]     25        :bidirect       cube-pins[1])
(cube-pins[2]     24        :bidirect       cube-pins[2])
(cube-pins[3]     23        :bidirect       cube-pins[3])
(cube-pins[4]     22        :bidirect       cube-pins[4])
(cube-pins[5]     21        :bidirect       cube-pins[5])
(M-pins[0]        20        :bidirect       M-pins[0])
(M-pins[1]        19        :bidirect       M-pins[1])
(vss              18        :vss            vss)
(vdd              17        :vdd            vdd)
(M-pins[2]        16        :bidirect       M-pins[2])
(M-pins[3]        15        :bidirect       M-pins[3])
(M-pins[4]        14        :bidirect       M-pins[4])
(M-pins[5]        13        :bidirect       M-pins[5])
(M-pins[6]        12        :bidirect       M-pins[6])
(M-pins[7]        11        :bidirect       M-pins[7])
(i-pins[0]        10        :bidirect       i-pins[0])
))
#|
```

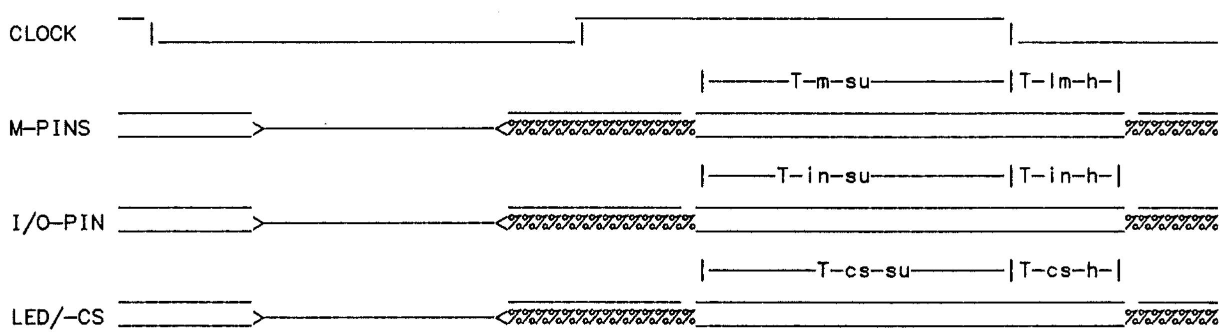
ALL CYCLES



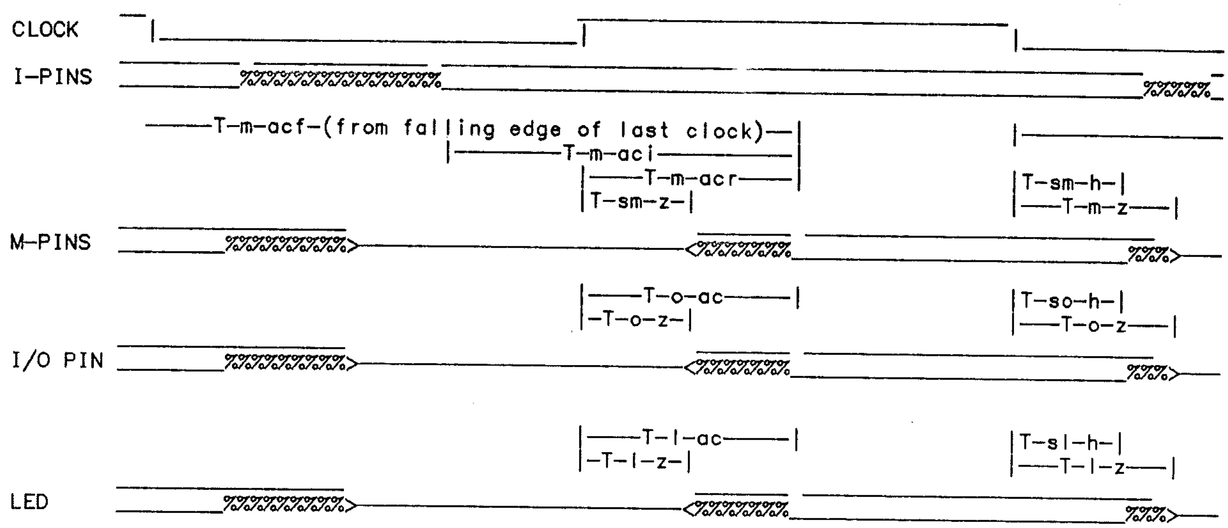
READ CYCLE



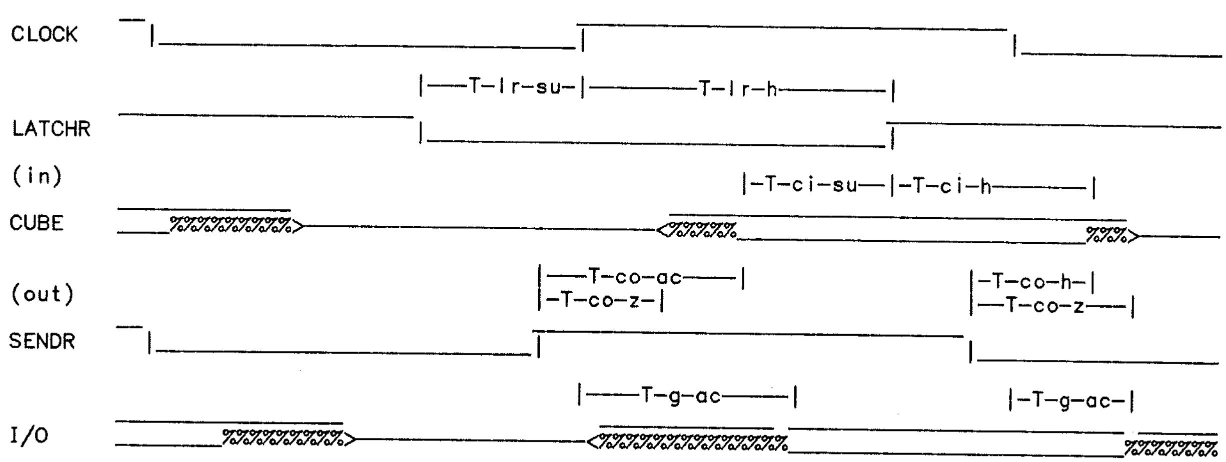
LOAD or RUG CYCLE



STORE CYCLE



ROUTE CYCLE



TIMING TABLE FOR THE CMCHIP

NAME	CYCLE	PIN	DESCRIPTION	NSEC.	
T-c-l		CLOCK	CLOCK LOW	50	MIN
T-c-h		CLOCK	CLOCK HIGH	50	MIN
T-op-su		OP-PINS	SETUP TIME	-2	MIN
T-op-h		OP-PINS	HOLD TIME	20	MIN
T-i-su		INSTRUCTION PINS	SETUP TIME	-2	MIN
T-i-h		INSTRUCTION PINS	HOLD TIME	20	MIN
T-e-ac		ERROR PIN	ACCESS TIME	30	MAX
T-e-h		ERROR PIN	HOLD TIME	8	MIN
T-m-su	LOAD	MEMORY PINS	SETUP TIME	-2	MIN
T-m-h	LOAD	MEMORY PINS	HOLD TIME	20	MIN
T-cs-su	LOAD	SC PIN	SETUP TIME	-2	MIN
T-cs-h	LOAD	SC PIN	HOLD TIME	20	MIN
T-in-su	LOAD	INPUT PIN	SETUP TIME	-2	MIN
T-in-h	LOAD	INPUT PIN	HOLD TIME	20	MIN
T-i-z	READ	INSTRUCTION PINS	TRI-STATE TIME	8	MIN
T-i-z	READ	INSTRUCTION PINS	TRI-STATE TIME	30	MAX
T-i-ac	READ	INSTRUCTION PINS	ACCESS TIME	30	MAX
T-ir-h	READ	INSTRUCTION PINS	HOLD TIME	8	MIN
T-mi-h	READ	MEMORY PINS	HOLD TIME	8	MIN
T-mi-ac	READ	MEMORY PINS	ACCESS TIME (ecc mode)	60	MAX
T-mi-ac	READ	MEMORY DATA PINS	ACCESS TIME (parity mode)	20	MAX
T-mi-ac	READ	MEMORY PARITY PIN	ACCESS TIME (parity mode)	46	MAX
T-cs-ac	READ	LED-CS PIN	ACCESS TIME (fall)	15	MAX
T-cs-ac	READ	LED-CS PIN	ACCESS TIME (rise)	8	MIN

T-m-acf	STORE	MEMORY PINS	ACCESS FROM FALL (ecc mode)	140	MAX
T-m-acf	STORE	MEMORY DATA PINS	ACCESS FROM FALL (parity mode)	75	MAX
T-m-acf	STORE	MEMORY PARITY PIN	ACCESS FROM FALL (parity mode)	95	MAX
T-m-acr	STORE	MEMORY PINS	ACCESS FROM RISE	45	MAX
T-m-aci	STORE	MEMORY PINS	ACCESS FROM INST. (ecc mode)	70	MAX
T-m-aci	STORE	MEMORY PINS	ACCESS FROM INST. (parity mode)	75	MAX
T-g-su	LOAD	GLOBAL	SETUP TIME		MIN
T-g-h	LOAD	GLOBAL	HOLD TIME		MIN
T-g-ac	STORE	GLOBAL	ACCESS TIME		MA%
T-g-h	STORE	GLOBAL	HOLD TIME		MIN
T-g-z	STORE	GLOBAL	TRI-STATE TIME		MIN
T-g-r	STORE	GLOBAL	RELEASE TIME		MA%
T-l-ac	STORE	LED	ACCESS TIME		MA%
T-l-h	STORE	LED	HOLD TIME		MIN
T-l-z	STORE	LED	TRI-STATE TIME		MIN
T-l-r	STORE	LED	RELEASE TIME		MA%

```
NOTE:  BETA EUNUCH simulated times can be found in:
       "a:>cm>beta>timing>eunuch-times.text".

Questions:

25. add I/O sequence to sequences section

29. off chip wires not protected by parity:
    OP
    News
    global
    error
    I/O
    msgp
    Direct cube writing
    Sendr latchr

30. Direct cube write sequence in sequences section looks wrong.

end
|#
```